2017

Proyecto : Metaheuristics for Logistics

MÉTODOS AVANZADOS DE RESOLUCIÓN DE PROBLEMAS

MOHAMMED MAHRACH

CONTENIDO

1. DESCRIPCIÓN DE LOS PROBLEMAS

1.1. Problemas

1.1.1. BPP (CONTAINER LOADING PROBLEM)

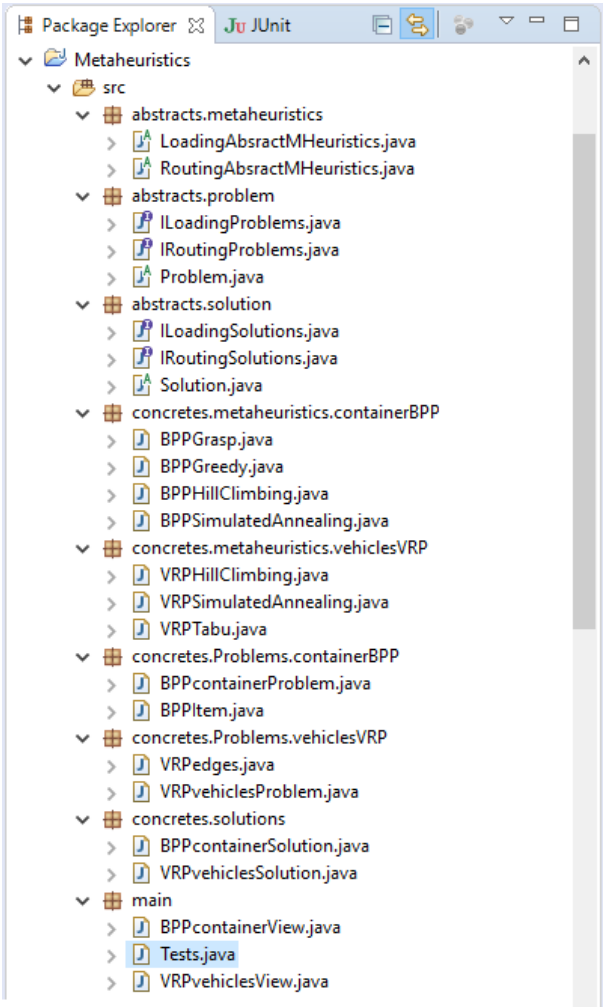
El problema de carga de un solo contenedor es uno de los problemas más difíciles en el corte y embalaje. Es un problema de optimización tridimensional en el que tenemos que empacar un conjunto de objetos rectangulares, en un objeto rectangular grande, contenedor, de tal manera que el empaque optimice algún criterio mientras satisface un conjunto de restricciones. Aunque en este caso solo trataremos de resolver este problema en una dimensión, considerando que una área del contenedor y objetos con área y prioridad.

1.1.2. VRP (VEHICLE ROUTING PROBLEM)

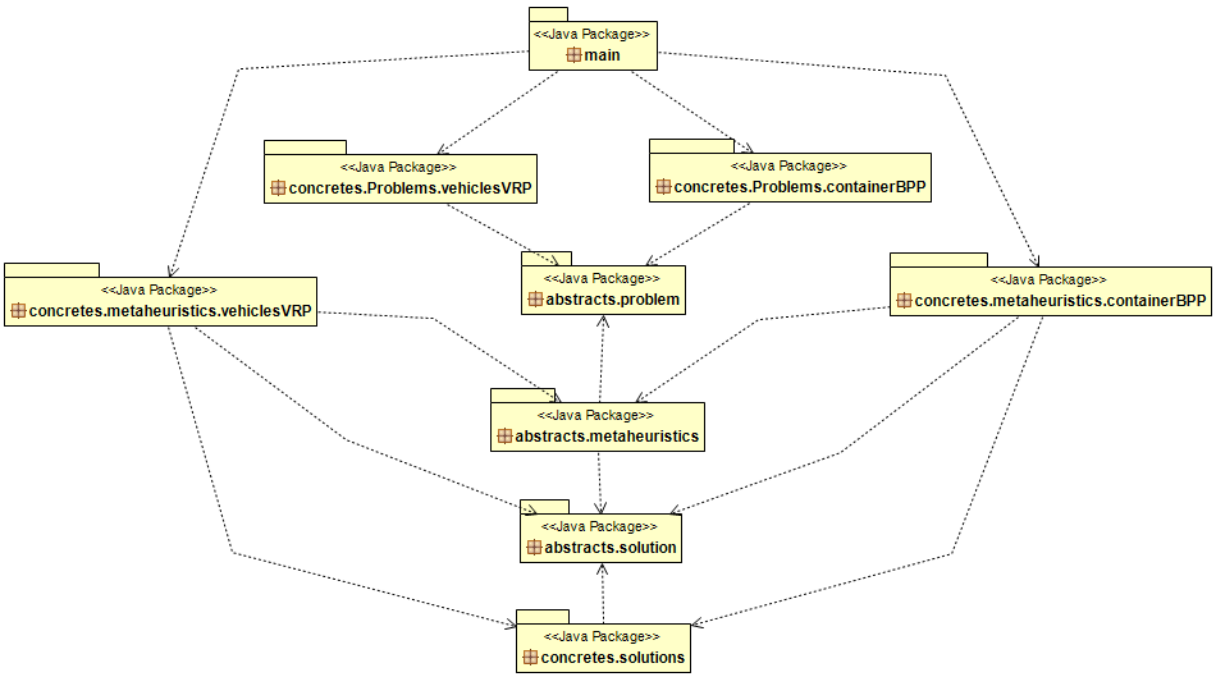
El problema de enrutamiento de vehículos (VRP) es un problema de optimización combinatoria de gran importancia en diferentes entornos logísticos, consiste en servir una serie de clientes ubicados geográficamente de manera dispersa, para atenderlos se cuenta con vehículos que parten desde un deposito central, el problema consiste en asignar a cada vehículo una ruta de clientes, de manera que se minimice el costo de transporte, en este caso solo contamos con un vehículo, será parecido al problema STP solo que la instancia del problema será un grafo doble dirigido con dos aristas entre dos vértices con pesos/costes diferente.

1.2. Desarrollo

1.2.1. Esquema general

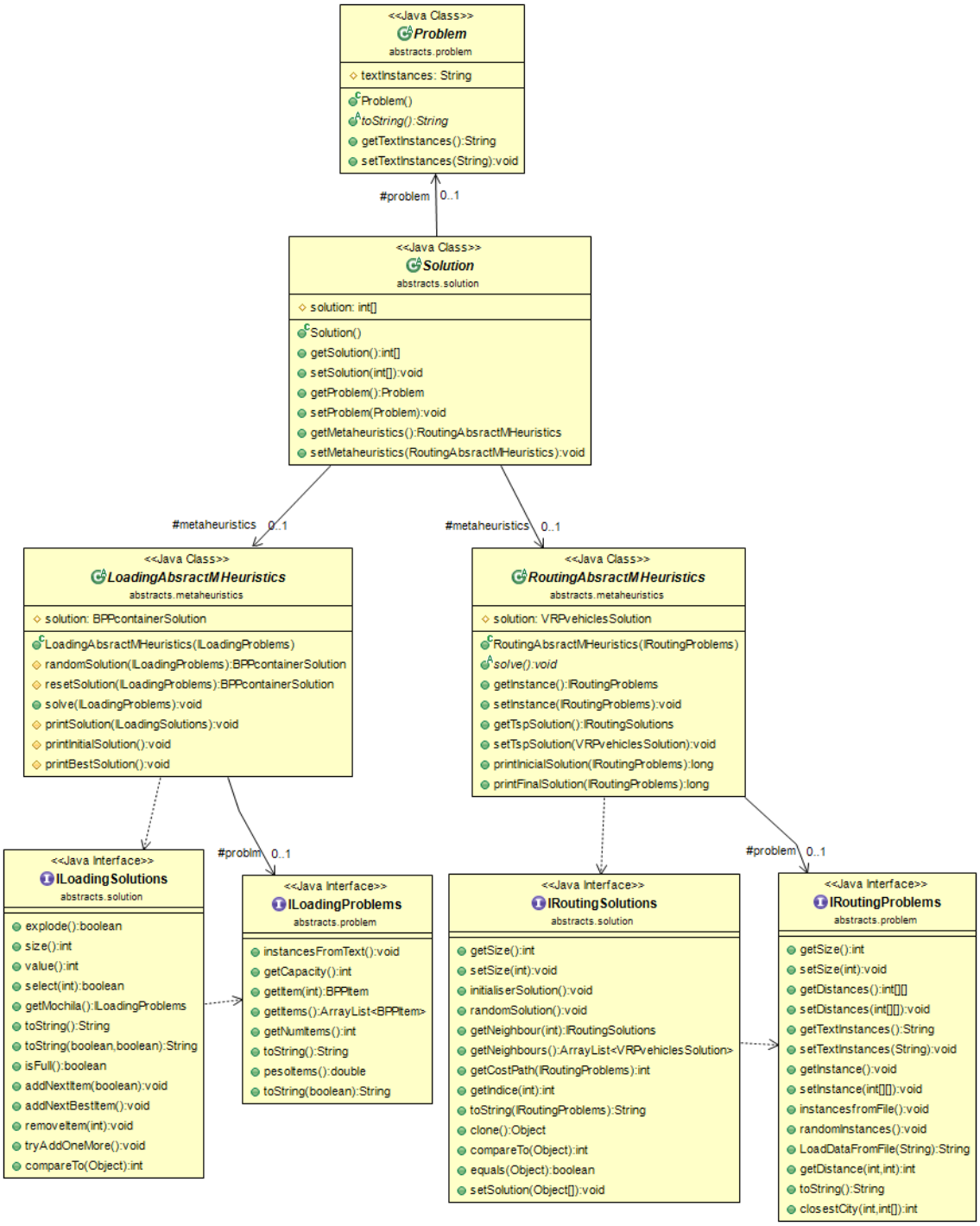


1.2.2. Diagrama de paquetes

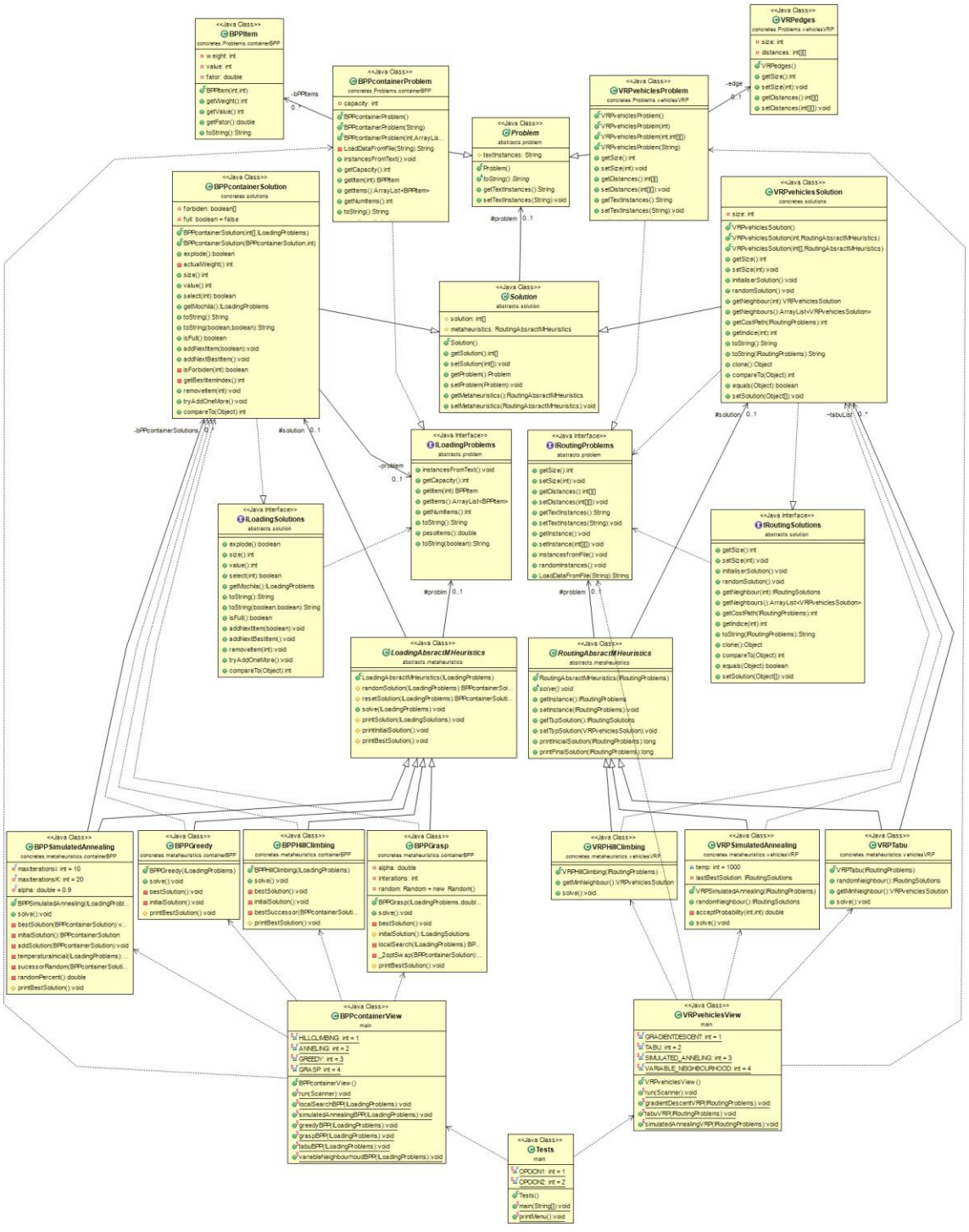


1.2.3. Abstract class

Se ha elaborado un diseño con la máxima extensibilidad tanto para que el código sea fácil de leer como para facilitar crecimiento futuro del código con menor esfuerzo, por lo cual se ha creado una estructura abstracta utilizando varios patrones de diseño donde se puede extender creando nuevos tipos de problemas (Rutas. Cargas. Localización,..), Nuevos problemas concretos (TSP, BPP,..) y nuevo Metaheurísticas, donde se puede ver en el siguiente diagrama:



1.2.4. Esquema completo del software



1.3. Algoritmos

1.3.1. Búsqueda local

Es un proceso iterativo de búsqueda en el que, dada una solución actual, se selecciona una solución de su entorno para continuar la búsqueda. Esta solución se selecciona si mejora a la actual, si no se desecha. En este caso se generará una solución aleatoria y a partir de ésta se irán generando vecinos y cambiando la solución si ésta mejora.

Utilizaré el algoritmo HillClimbing con dos variantes de este tipo algoritmos para cada problema:

- FirstImprovement: Para BPP (CONTAINER LOADING PROBLEM) En este caso el algoritmo para de buscar en el momento en el que encuentra una solución que mejore a la que ya se tiene.
- BestImprovement: Para VRP (VEHICLE ROUTING PROBLEM), El algoritmo explorará todas las posibles soluciones vecinas y seleccionará la mejor de entre todas ellas.

1.3.2. simulated annealing (Enfriamiento Simulado)

Es un algoritmo de búsqueda por entornos con un criterio probabilístico de aceptación de soluciones basado en termodinámica. Permite que algunos movimientos sean hacia soluciones peores, para evitar que se quede en óptimos locales. Hay que controlar estos movimientos, ya que nos pueden desviar del óptimo global. Para ello siempre se guarda la mejor solución hasta el momento. Conforme va avanzando la búsqueda va disminuyendo la probabilidad de que salte a una solución peor que la actual. Se usa una probabilidad de aceptación de nuevas soluciones peores que es función exponencial de la modificación de la función objetivo.

Otras metaheurísticas simplemente reducen o incrementan esta probabilidad de aceptar empeoramientos para modular la exploración y explotación de la búsqueda

1.3.3. Busqueda Tabú

Es una metaheurísticas de búsqueda con memoria trata de utilizar la memoria del proceso de búsqueda para mejorar su rendimiento.

En el origen del método el propósito era sólo evitar la reiteración en una misma zona de búsqueda recordando las últimas soluciones recorridas; memoria a corto plazo.

La forma más directa de introducir la memoria del proceso de búsqueda en el procedimiento de búsqueda no monótono es considerar una función de aceptación que tenga en cuenta la historia de la búsqueda, generalmente con una lista Tabú:

1.3.4. Greedy (búsqueda por entorno monótona aleatoria)

La metaheurística básica de búsqueda por entorno monótona aleatoria consiste en seleccionar iterativamente una solución al azar del entorno de la solución actual que es sustituida por ésta si se produce una mejora (el mejor primero).

La solución de partida se puede obtener por cualquier procedimiento arbitrario y el criterio de parada reflejará el estancamiento de la búsqueda en un mínimo local presumible cuando en un cierto número de intentos no se pueda mejorar la solución actual.

Las metaheurísticas intensifican la búsqueda en torno a cada solución actual seleccionando el mejor entre una serie de soluciones del entorno, cada vez de mayor tamaño, la intensidad de la búsqueda viene determinada por el número o la proporción de soluciones vecinas de la solución actual entre las que se toma la mejor.

1.3.5. GRASP Greedy Randomized Adaptive Search Procedure

El procedimiento de búsqueda adaptativa aleatoria codicioso (también conocido como GRASP) es un algoritmo metaheurístico comúnmente aplicado a problemas de optimización combinatoria. GRASP por lo general consiste en iteraciones a partir de sucesivas construcciones de una solución greedy aleatoria y mejoras iterativas de la misma a través de una búsqueda local. Las soluciones greedy aleatorias se generan añadiendo elementos al conjunto de soluciones del problema a partir de una lista de elementos clasificados por una función greedy según la calidad de la solución que lograrán. Para obtener variabilidad en el conjunto candidato de soluciones greedy, los elementos candidatos bien clasificados se colocan a menudo en una lista restringida de candidatos (también conocida como RCL) y se eligen al azar al construir la solución.

1.3.6. Algoritmos implementados

	PROBLEMS	
METAHEURISTIC	VRP (VEHICLE ROUTING)	BPP (CONTAINER LOADING)
HILL CLIMBING	✓	✓
HILL CLIMBING Modificado	✓	x
SIMULATED ANNEALING	✓	✓
TABU	✓	x
GREEDY	x	✓
GRASP	x	✓

2. DESCRIPCIÓN DEL ESQUEMA DE REPRESENTACIÓN

2.1. BPP (CONTAINER LOADING PROBLEM)

Para este problema se han utilizado dos estructuras, una para almacenar la instancia a utilizar y otra para almacenar la solución.

- La estructura de la instancia se compone por un entero que almacenará el tamaño de la matriz de distancias, el número de elementos que se van a seleccionar y de la matriz de distancias.
- La estructura de la solución se compone de un vector booleano que indica qué elementos se han seleccionado.

2.2. VRP (VEHICLE ROUTING PROBLEM)

Para este problema he utilizado dos estructuras, una para almacenar la instancia a utilizar y otra para almacenar la solución.

- La estructura de la instancia es objeto de la clase Edges “matice de las distancias de aristas” este objeto se compone por un entero que almacenará el tamaño y de la matriz de distancias.
- La estructura de la solución se compone de un vector donde se almacena el orden de los nodos para la suma de las distancias entre ellos.

3. DESCRIPCIÓN DE LA ESTRUCTURA DE LOS MÉTODOS DE BÚSQUEDA

3.1. Búsqueda Local (HillClimbing)

3.1.1. BPP (CONTAINER LOADING PROBLEM) firstImprovement

```
do {
	this.solution = this.randomSolution(lProblem);
	} while(this.solution.full());
}

while(true){

	BPPSolution neighbor = this.getMinNeighbour(solution);
	if(neighbor.priority() <= solution.priority())
		break;
	solution = neighbor;
}
```

3.1.2. VRP VEHICLE ROUTING PROBLEM *bestImprovement*

```
do {
	VRPSolution neighbor = this.getBestNeighbour();
	if(neighbor.getCostPath() - solution.getCostPath() < 0) {
		solution = neighbor;
	} else break;
} while(true);
```

3.2. Enfriamiento Simulado

3.2.1. BPP (CONTAINER LOADING PROBLEM)

```
BPPSolution newSolution = null;
double temperature = this.temperaturaInicial(this.lProblem);
int k = 0;
do {
	int i = 0;
	do {
		newSolution = this.sucessorRandom( initialSolution );
		int variance = newSolution.priority() - initialSolution.priority();

		if( variance >= 0 && !newSolution.full() ){
			initialSolution = newSolution;
			this.addSolution( initialSolution );
		} else {
			double rand = this.randomPercent();
			double pVariance = -((double)variance / temperature);

		if( pVariance > rand && !newSolution.full() ){
				initialSolution = newSolution;
				this.addSolution( initialSolution );
			}
		}
		i++;
	} while( i < this.maxIterationsI );

	k++;
temperature = this.alpha * temperature;
} while( k < this.maxIterationsK );
Return printBestSolution();
```

3.2.2. VRP VEHICLE ROUTING PROBLEM

```
Temp = 1000;
do {
	neighbourSolution = this.randomNeighbour();
	rand = new Random().nextDouble();
	if( rand < acceptedProb(solution.costPath(),neighbourSol.costPath()) ){
if( neighbourSolution.getCostPath() < lastBestSolution.getCostPath())
		lastBestSol = neighbourSolution;
		solution = neighbourSolution;
		}
	temp--;
} while( temp > 1 );

if( solution.getCostPath(problem) > lastBestSolution.getCostPath(problem))
solution = lastBestSolution;
```

3.3. Tabu

3.3.1. VRP VEHICLE ROUTING PROBLEM

```
int cmpt = this.getInstance().getSize()*10;
VRPvehiclesSolution s1 = solution, newSolution;
this.tabuList.add(s1);
do {
	newSolution = this.getMinNeighbour();
	if(newSolution.compareTo(s1) < 0) {
		solution = newSolution;
		this.tabuList.add(newSolution);
	}
	else cmpt--;
} while(cmpt>0);
```

3.4. Greedy

3.4.1. BPP (CONTAINER LOADING PROBLEM)

```
int size = Problem.getNumItems();
		int[] sol = new int[size];
		for( int i : sol )
			sol[i] = 0;

while( ! solution.isFull() ){
	int index = getBestItemIndex();
	for (int i = 0; i < solution.length; i++)
		if (this.problem.getItem(i).getFator() > bestFator) {
			index = i;
			bestFator = this.problem.getItem(i).getFator();
		}
	}


	if (index != -1) {
		this.solution[index] = 1;
		if (this.full()) {
			this.solution[index] = 0;
		}
	} else
		this.full = true;
}
```

3.5 Grasp

3.5.1. BPP (CONTAINER LOADING PROBLEM)

```
Solution alternativeSolution;
alternativeSolution = localSearch();
if( alternativeSolution.priority() > solution.priority()){
	solution = alternativeSolution;
}
```

3.2. Generación de vecinos

Para este algoritmo se han implementado dos formas diferentes, una por cada tipo de problema. En ambas la única estructura que se mantiene es la de la solución, que en cada caso es la descrita para cada problema en el punto anterior.

3.2.1. BPP (CONTAINER LOADING PROBLEM)

```
	private Solution getMinNeighbour(BPPcontainerSolution actualSolution){

		BPPcontainerSolution newSolution = null;
		BPPcontainerSolution bestSolution = actualSolution;
		int bestValue = bestSolution.priority();
		int length = actualSolution.getProblemInstance().getNumItems();

		for(int i = 0; i < length; i++){
			if(!actualSolution.select(i)){
				newSolution = new containerSolution(actualSolution, i);
				int newValue = newSolution.priority();
				if(newValue >= bestValue && !newSolution.full()){
					bestValue = newValue;
					bestSolution = newSolution;
				}
			}
		}

		return bestSolution;
	  }
```

3.2.2. VRP VEHICLE ROUTING PROBLEM

```
  public ArrayList<VRPSolution> getNeighbours(){
	ArrayList<VRPSolution> neighbours = new ArrayList<VRPSolution>();
	for( int i = 0 ; i < size-1 ; i++) {
		neighbours.add(this.getNeighbour(i));
	}
	return Collections.min(neighbours);
   }
```

4. EXPERIMENTOS Y ANÁLISIS DE RESULTADOS

4.1. Descripción de las instancias

4.1.1. BPP (CONTAINER LOADING PROBLEM)

He realizado dos tipos de instancias El primero por entrada de datos por fichero y el otro la generación aleatoria de instancias.

- El nombre de las instancias por entrada de datos está compuesto por un código alfanumérico compuesto por el nombre el problema y un número de la instancia (VRP_x o BPP_x).
- Las instancias que se generan aleatoriamente se muestran por pantalla antes de seleccionar la metaheurística de solución del problema.

En la instancia encontramos en la primera fila el número de nodos del grafo seguido del número de nodos a elegir para calcular la distancia. Seguidamente encontramos por filas el número de un nodo, el número de un nodo diferente y la distancia que hay entre estos dos nodos.

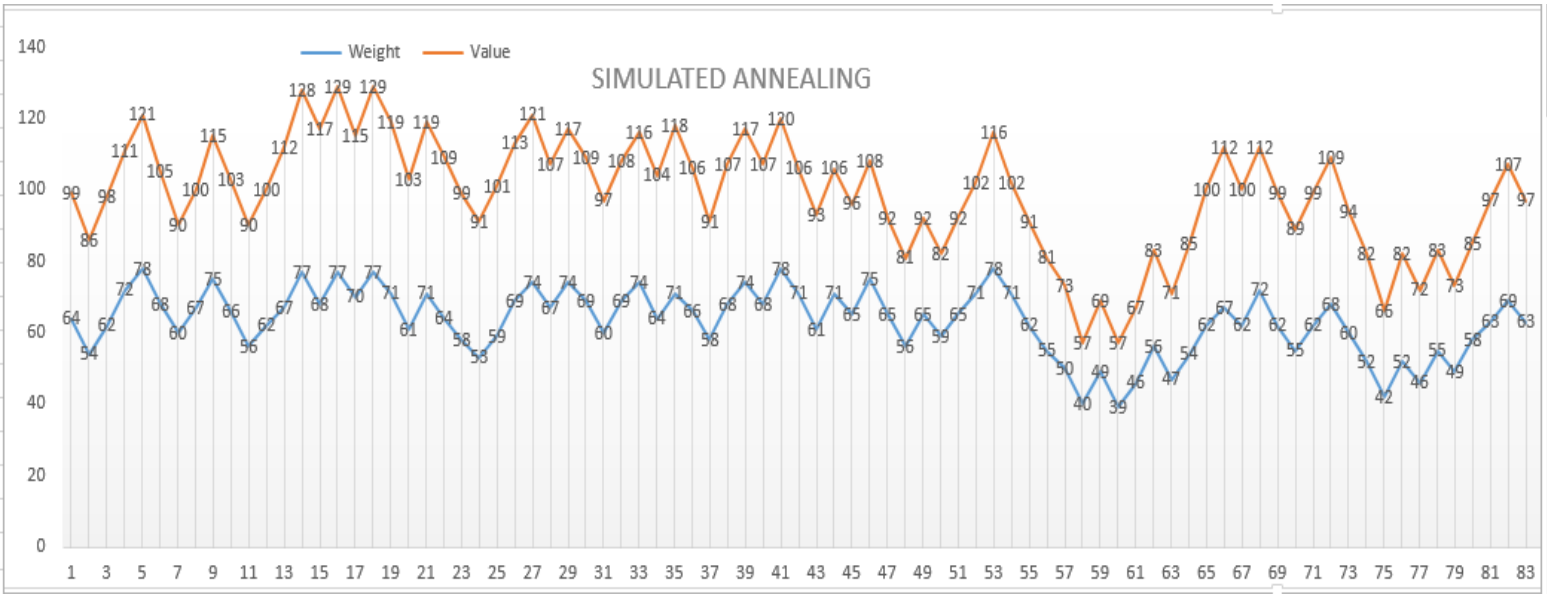
4.1.2. VRP (VEHICLE ROUTING PROBLEM)

Estas instancias contienen en la primera fila el número de ciudades comprendidas en el grafo seguidas de las distancias entre las ciudades. En cada fila están las distancias de la ciudad que corresponde a esa posición (si es la primera fila, la primera ciudad) con las demás ciudades, separadas por un espacio. Los nombres de las instancias están compuestos por unas letras y un número, que indica el número de nodos del grafo, excepto la instancia “p01.txt”, que tiene 15 ciudades.

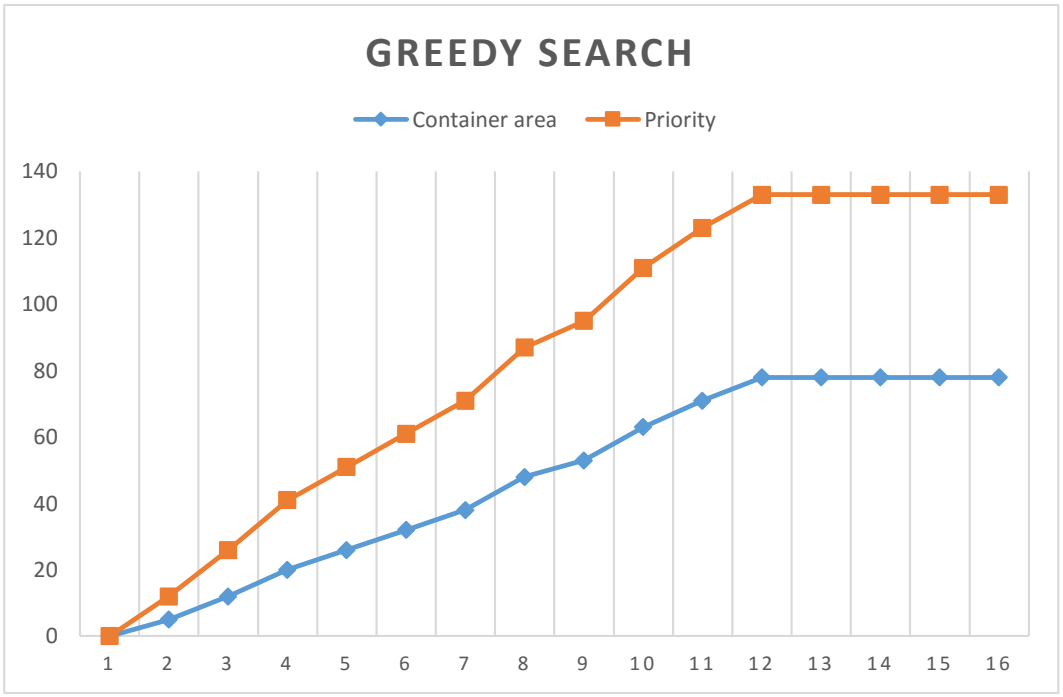
4.2. Resultados obtenidos

4.2.1. BPP (CONTAINER LOADING PROBLEM)

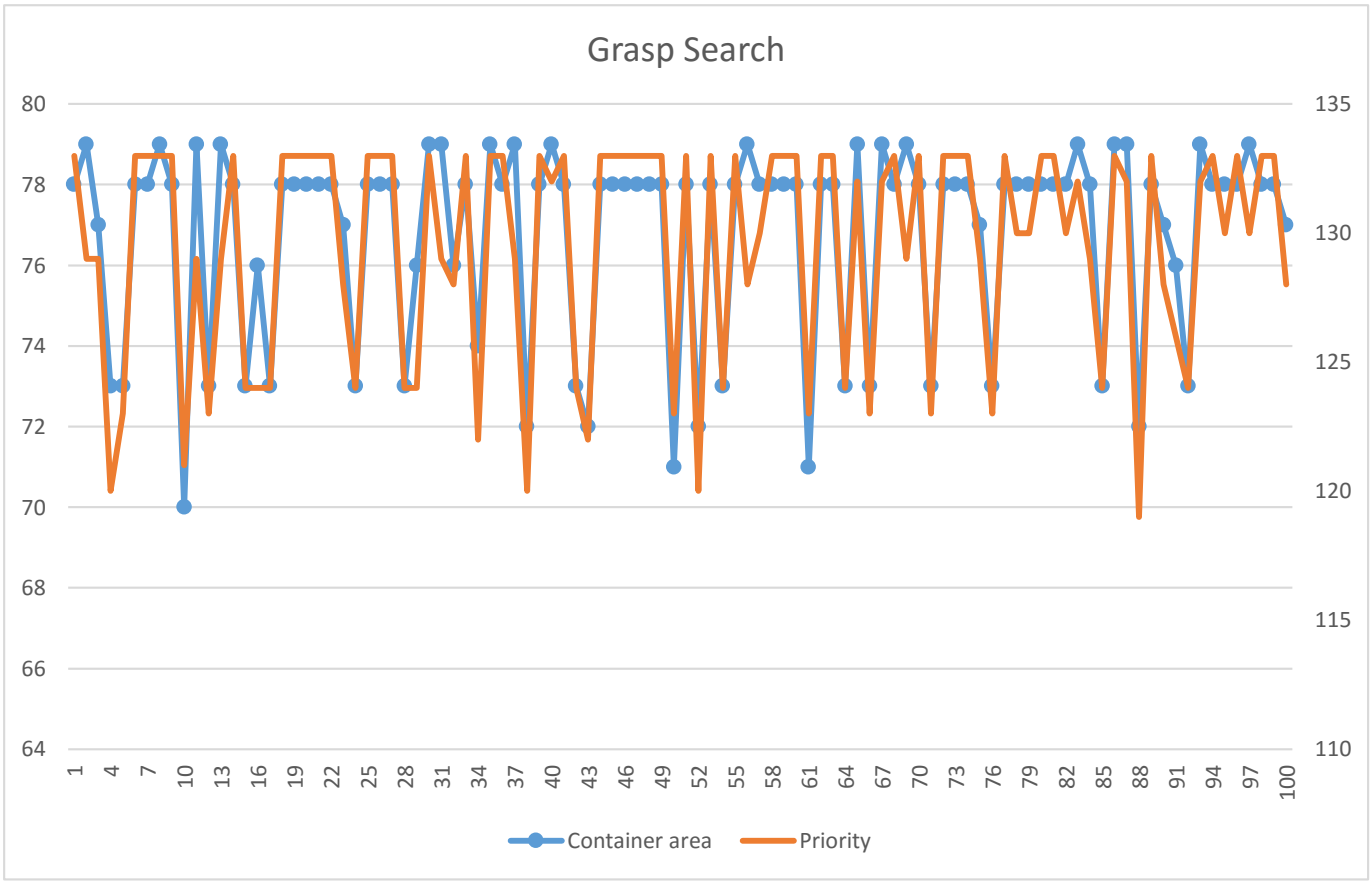
Simulated anneling



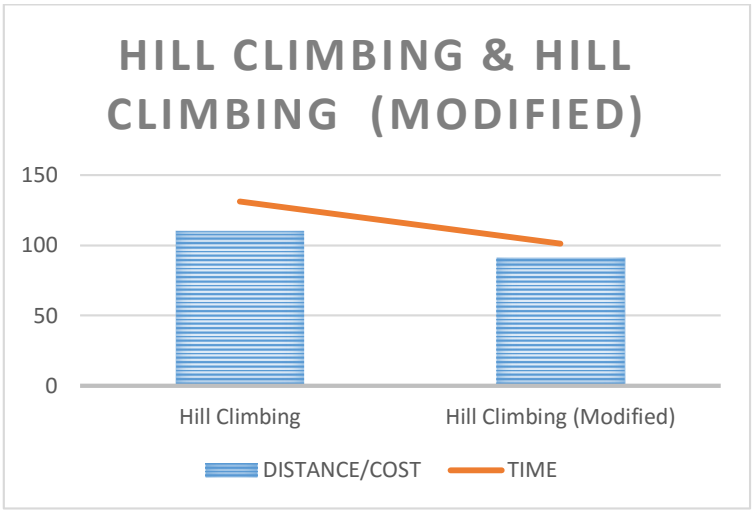
4.2.2. Greedy



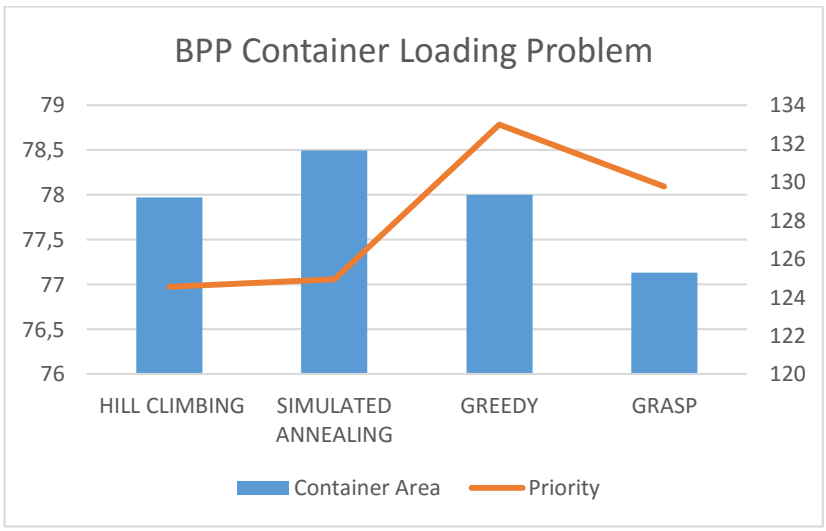
4.2.3. Grasp



Se ha realizado una versión modificada del hill Climbing para lograr una ligera mejora en tiempo y distancia entre las dos versiones

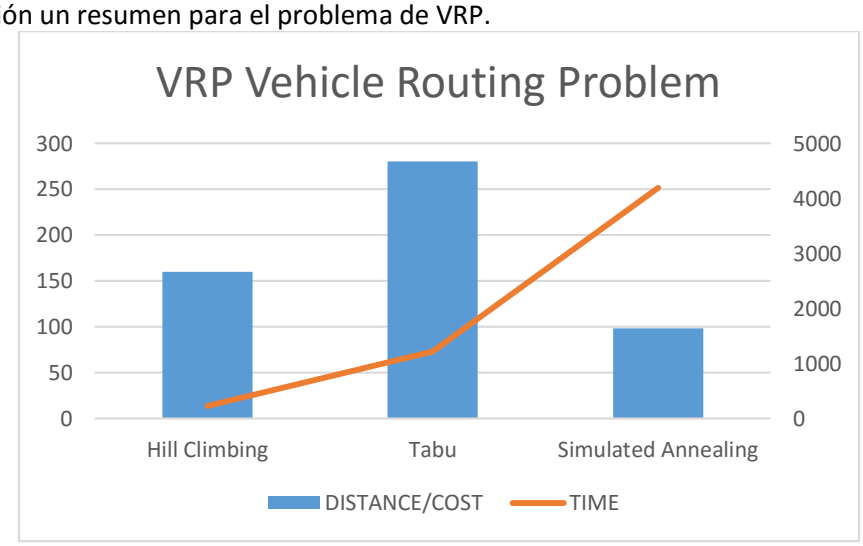


Un resumen entre las 4 metahueristicas se muestra a continuación donde podemos comparar los tiempo de ejecución y el área del contenedor optimizada para cada metaheurística



4.2.2. VRP (VEHICLE ROUTING PROBLEM)

Y a continuación un resumen para el problema de VRP.



4.3. Análisis de los resultados

4.3.1. BPP (CONTAINER LOADING PROBLEM)

Observamos el grafico que muestra los resultados referentes a la aplicación de los algoritmos al problema VRP VEHICLE ROUTING PROBLEM. Estos resultados son la media de la aplicación de los algoritmos con tres semillas.

Como se puede observar, la metaheurística **Simulated anneling ha sido el mejor en optimizar el tamaño del contenedor** pero con paquetes de menos prioridad, sin embargo notamos que Greedy ha optimizado la prioridad de los paquetes y en optimizar el tamaño del contenedor ha quedado segundo así podría ser un candidato en término medio.

4.3.2. VRP VEHICLE ROUTING PROBLEM

Observamos el grafico que muestra los resultados referentes a la aplicación de los algoritmos al problema VRP VEHICLE ROUTING PROBLEM. Al contrario que en los resultados con BPP (CONTAINER LOADING PROBLEM),

En este caso **el peor de los resultados lo tiene el algoritmo Tabú** ya que nos proporciona soluciones (media de soluciones) de mayor coste (distancia entre nodos) y ha tardado bastante en ejecutarse, pero no tardó tanto como el **Simulated anneling, este último gana de nuevo el mejor puesto en optimizar las distancias de las rutas** y el peor puesto en tiempo de ejecución, esto podría deberse (en los dos problemas) a que la temperatura desciende más lenta de lo necesario así que le da tiempo optimizar la solución del problema comprobando casi todas las posibles soluciones por un lado, pero tardando demasiado en tiempo de ejecución, en este caso me quedaría con **HillClimbing** ya que aunque no tiene la mejor solución pero la ejecutó en un tiempo mucho más corto que el **Simulated anneling**.
Puede que estas diferencias se deben a que el problema a tratar es diferente, ya que en uno queremos la mínima distancia que conecte a todos los nodos y en el otro caso queremos maximizar el número de paquetes en un contenedor y los de mayor prioridad.
Finalmente podemos concluir que las diferente metaheurística se comportan de forma diferente para cada problema y no podemos concluir que hay una metaheurística mejor que otra en términos absolutos, pero si podemos elegir una metaheurística u otra depende de varios factores del problema, el objetivo que tengamos (tiempo, tamaño de la muestra, etc.,.).